Assessing Defragmentation Strategies for FPGAs

Manuel G. Gericota, Gustavo R. Alves,
Luís F. Lemos
DEE/ISEP
Rua Dr. Antº Bernardino de Almeida
4200-072 Porto
PORTUGAL
{mgg, gca, lfl}@isep.ipp.pt

José M. Ferreira
DEEC/FEUP
Rua Dr. Roberto Frias,
4200-465 Porto
PORTUGAL
jmf@fe.up.pt

Abstract

Fragmentation on dynamically reconfigurable FPGAs is currently a major obstacle to the efficient management of its logic space. When resource allocation decisions have to be made at run-time a relocation of currently running functions may be necessary to release enough contiguous resources to implement incoming functions.

Relocation should take into account any specifics of function's functionality and also those of the FPGA's architecture as to not affect system's performance. A simple and fast method to assess performance degradation of a function during relocation is proposed in this paper. This method is based on previous function labelling and on the concept of proximity vectors.

1. Introduction

Due to their growing densities and surpassing flexibility, Field Programmable Gate Arrays (FPGAs) are rapidly proving to be a cost-effective alternative to the increasing cost and risk of designing with ASICs. Moreover, the reductions in reconfiguration times and the new features introduced, such as run-time partial reconfiguration and self-reconfiguration, made possible the implementation of the concept of virtual hardware defined in the early 1990s: the hardware resources are supposed to be unlimited and implementations that oversize the reconfigurable logic space available are resolved by temporal partitioning [1].

Generally, an application comprises a set of functions that are predominantly executed in sequence, or with a low degree of parallelism, in which case their simultaneous availability is not required. Functions may be swapped in real time becoming operational only when needed and being substituted if their availability is no longer required. In this way, it becomes feasible to use a single device to run an application that in total requires far more than 100% of the FPGA available resources, by swapping functions in and out of the FPGA as needed. Furthermore, as density increases and bigger FPGAs become available, several applications may even share spatial and temporally the same reconfigurable logic space.

When the logic space of an FPGA is shared among several functions belonging to a number of different applications, each with its own requirements in spatial and temporal terms, fragmentation of the logic space may occur [2]. Being composed by an array of identical Configurable Logic Blocks (CLBs) it may seem at first sight that the problem may be regarded as a two-dimensional one. However, it will be shown that several other factors have to be considered when relocating a function during run-time. Decision time is also of concern since defragmentation procedures are usually time costly, which may threaten the viability of sharing the FPGA's logic space. New incoming functions may have to wait a long time before having enough contiguous logic space available to be implemented, decreasing the whole system's performance below acceptable levels, or even disrupting its operation.

A new method to quickly evaluate relocation decisions is proposed in this paper.

2. Formal approaches

When the logic space is shared by multiple independent functions, each requiring a different amount of resources to run efficiently, as the resources are allocated to functions and later released, many small areas of free resources are created. These unallocated portions tend to become so small that they fail to satisfy any request and therefore remain unused - the FPGA logic space gets fragmented [3]. Suitable arrangements can be

• This work is supported by an FCT program under contract POSC/EEA-ESE/55680/2004

designed if the requirements of functions and their sequence are known in advance, but not when placement decisions have to be made at run-time [4].

The solution to the problem is to consolidate unused areas within the FPGA without halting the operation of currently running functions. If a new function cannot be allocated immediately due to lack of contiguous free resources, a suitable rearrangement of a subset of the executing functions must be implemented to overcome the problem.

Some authors treated defragmentation as a strictly packing problem [4-6]. Despite the undeniable interest of this approach, the functionality of the functions that are being relocated and the resources they require to run efficiently must be taken into account, otherwise, a degradation of the system's performance or even its disruption may occur. These formal methods also present another drawback: the long time they usually take to reach a solution, which makes run-time defragmention impracticable.

Other authors presented some strategies to avoid these pitfalls considering that functions have fixed-shapes, which removes flexibility to the solutions [7], or imposing initial fixed logic space areas, like pagination in computer design, which, despite solving possible external fragmentation problems, tends to create internal fragmentation [8, 9]. Moreover, imposing fixed shapes or areas implies a set of floorplanning constraints that may severely limit function's performance.

In general, there is a tendency to model the FPGA as a regular array structure. While this assertion may be true regarding the CLBs position inside the array, it is inaccurate when the routing infrastructure is considered. This irregularity is mainly due to the presence of dedicated routing resources available to enhance specific applications, like counters and adders, which have a tremendous impact on function's performance.

Notice that accessing the reconfiguration mechanism is independent from the operation of the functions being executed in the FPGA. Therefore, defragmentation may be implemented as a background process, running concurrently with the operation of currently implemented functions, without disturbing or impaired them, and not only when a new incoming function is claiming area to be implemented. As a result, waiting times will be reduced and the overall system's performance improved, as defragmentation can be a highly time-consuming task. A metric to determine when to perform defragmentation is proposed in [10].

From the analysis of previous proposals it is possible to identify some key problems:

1. The lack of shape flexibility, which restricts defragmentation performance;
2. The defragmentation algorithms are too complex to be performed in real-time, making run-time defragmentation impracticable;
3. The specific architecture of the FPGA is not taken into account; hence changes on function's performance as a result of relocation are not evaluated or even considered.

The proposed methodology, described in the next sections, contemplates all these problems.

3. Function tags

To enhance the performance of specific types of functions, FPGA's architectures present some special features, like dedicated carry lines to increase speed on arithmetic functions (e.g. counters or adders). In the architecture of Virtex FPGAs from Xilinx, which are being used in this research work, these lines span the FPGA vertically, enabling only the interconnection of vertically adjacent CLBs. The use of dedicated carry lines, with very low propagation delays (in the order of a few picoseconds), enabled during our experiments with a 24-bit binary counter to reach frequencies of operation of around 145 MHz using a XCV200. However, the maximum frequency of operation of this counter decreases dramatically if one or more of this dedicated carry lines are substituted by generic interconnection resources. Figure 1 presents a comparison between the number of dedicated carry lines broken when the CLBs are reallocated horizontally and the decrease on the maximum frequency of operation in percentage terms. From this example it becomes obvious that it is mandatory for any defragmentation procedure to take both the FPGA's architecture and the functionality of the function to be relocated into consideration. Moreover, if the function is active, i. e. if the function is currently being used by an application, dynamic relocation techniques, as those described in [2], must be applied during the defragmentation procedure, otherwise its operation will be brought to a temporary halt, which may disrupt the operation of the whole system. The relocation of the function must be performed keeping as much as possible the vertical orientation of the function's placement. Besides, no more than one of the dedicated carry lines linking vertically adjacent CLBs should be broken during it. This means that only one adjacent CLB may be relocated at a time and that vertical adjacency must not be lost. Otherwise, the decrease on the maximum frequency of operation will be significant, as shown in figure 1, and may even compromise the correct operation of the system.

These two pieces of information, verticality and adjacency, are essential for the system to efficiently conduct defragmentation and may be attached as a tag to the function's configuration file, stored therein. When the file is retrieved and transferred into the FPGA, the processor responsible for the management of the logic space simply reads and stores this information and uses it if the need to relocate the function arises.

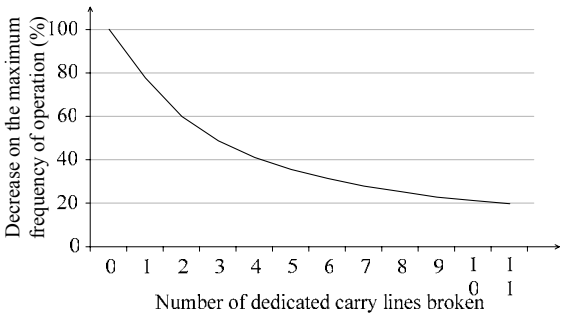

Fig. 1. Performance degradation

To evaluate the impact that changes in the shape and in the relative position of CLBs have in different functions, the same type of experiments were performed over a subset of the ITC'99 benchmark circuits – B01 to B14 [11]. The objective was to determine which parameters are involved in the performance degradation of particular functions to be able to formulate a simple set of rules to be used by the processor responsible for the logic space management to assess and guide the defragmentation procedure.

The experiments were conducted displacing vertically and horizontally each one of the functions and changing its relative shape, from a square-like shape to a rectangular one and rotating it 90º. These stressing conditions helped to put into evidence which parameters most affect performance degradation when functions are moved around. The results of the experiments are summarised in table 1.

Circuits B04, B05, B07, B11, B13 and B14 experienced considerable performance degradation when the shift of the whole function was carried out horizontally. Faster performance degradation arose when their spatial orientation tended to pass from the original vertical to a horizontal one. This occurs because all these functions make use of dedicated carry lines on their implementation. This conclusion confirmed the high importance of keeping unbroken the dedicated carry lines used by some functions.

Some functions, like the B11, comprise hundreds of gates but have a reduced number of carry lines. In this case, it is necessary to have a simple method to quickly identify the columns where these lines stand. Otherwise, the ability to reshape the function during a defragmentation procedure will be heavily constrained. In this case, the tag attached to the function configuration file must indicate the relative position inside the function of the column that must be kept unshaped.

On circuits B04, B05, B07, B11, B13 and B14 it was rather easy to identify the cause of the performance degradation when circuits were reshaped from its vertical orientation to a horizontal one. For the remaining circuits a deeper analysis of its implementation and functionality was needed to understand the origins of performance degradation.

Circuit reference	Number of occupied CLBs	Variation in the maximum frequency of operation (%)	
		Vertical shift	Horizontal shift
B01	6	-5,5	0,0
B02	1	0,0	0,0
B03	11	-1,9	-4,9
B04	54	-6,1	-29,3
B05	103	-17,3	-36,9
B06	5	-2,7	0,0
B07	31	-23,6	-37,8
B08	17	-5,8	-5,8
B09	12	-1,8	-4,9
B10	20	-7,5	-7,6
B11	39	-10,5	-36,0
B12	119	0,0	-1,2
B13	37	-4,3	-42,8
B14	333	-13,5	-47,8

Table 1. Evaluation of function's performance degradation with reshaping

4. Assessing relocation impact

The circuit B01 exhibits a different behaviour. Horizontal shifts do not degrade its performance, probably because it uses no carry lines. However, vertical shifts most decrease its maximum frequency of operation. In figure 2 it is shown how this circuit was placed inside a XCV200 FPGA by the design tools. The most noticeable aspect is the great number of lines that leave the CLBs located in the central column. In fact, two output signals in the upper CLB and one output signal in the lower CLB, whose nets are highlighted in the figure, drive a great number of inputs. To reduce propagation delays these CLBs were positioned by the design tools in the centre of the function's floorplan. If the central location of these two CLBs is changed, propagation delays will increase and the maximum frequency of operation of this function will decrease. This hypothesis was verified rotating the function 90º and relocating it in only one CLB column.

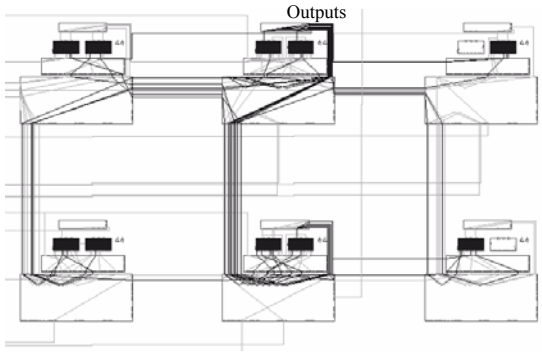

Fig. 2. Placement of circuit B01 inside a XCV200

The systematisation of the analysis led to the development of a new approach able to assess the impact relocation of CLBs with output signals driving a great number of inputs have over functions' performance: the application of the

concept of "proximity vectors", a vector associated to each interconnection and linking the CLB source to the CLB destination. The length of each vector, called proximity factor, is expressed in CLB units and calculated as the modulus of the distance between the CLB source and the CLB destination. This vector is expressed by:

$$\vec{f}_{px} = (r, c) \qquad \text{(eq. 1)}$$

where:

r = CLB destination row – CLB source row
c = CLB destination column – CLB source column

and the length of the vector is given by:

$$\left| f_{px} \right| = \sqrt{r^2 + c^2} \qquad \text{(eq. 2)}$$

A CLB with an output driving four different inputs will have associated to each one of the interconnections a vector, as exemplified in figure 3. Notice that if a CLB output is fed back to one of its inputs, the vector length will be zero. Minimising the sum of all proximity vectors of one CLB output results in the minimisation of the proximity factor associated to that output. This corresponds, for a given output and in terms of propagation delay, to the best position of that CLB inside the function.

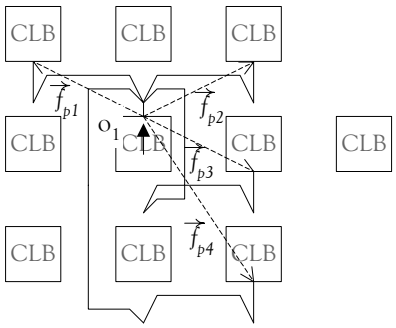

Fig. 3. Application example

When relocating the CLB, if the proximity factor increases performance degradation will occur. Generically, we can say that minimising each one of the output proximity factors of a function results in the minimisation of its global proximity factor, which corresponds to the best performance achievement, in terms of maximum frequency of operation.

The application of this concept to the remaining circuits has shown a consistent reproduction of results, confirming the initial hypothesis.

5. Conclusions

This approach has some advantages like:

1. It can be easily automated and integrated in current design tools;
2. The necessary computation time will be low compared with the temporal reanalysis of the whole function, even if a sole output drives one hundred inputs, as happens with circuit B12;
3. There is no need to perform a complete analysis of the function's performance after each CLB relocation, because, if the minimisation of the global proximity factor of the CLB is assured, the minimisation of the global proximity factor of the function will be assured, and therefore no performance degradation will occur.

All these factors enable its use at run-time to quickly and reliably assess and guide the strategy used to manage the defragmentation procedure.

Further work is being done to fine-tune the approach and to identify other possible specific sources of performance degradation, without increasing the complexity of the analysis. Otherwise, run-time defragmentation will be compromised.

References

[1] X.-P. Ling, H. Amano, "WASMII: a Data Driven Computer on a Virtual Hardware", *Proc. 1st IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 33-42.

[2] M. G. Gericota, G. R. Alves, M. L. Silva, J. M. Ferreira, "Run-Time Defragmentation for Dynamically Reconfigurable Hardware", in: *New Algorithms, Architectures and Applications for Reconfigurable Computing*. Springer, April 2005. pp. 117-129.

[3] M. Vasilko, "DYNASTY: A Temporal Floorplanning Based CAD Framework for Dynamically Reconfigurable Logic Systems", *Proc. 9th Intl. Workshop on Field-Programmable Logic and Applications*, 1999, pp.124-133.

[4] J. Teich, S. Fekete, J. Schepers, "Compile-time optimization of dynamic hardware reconfigurations", *Proc. Intl. Conf. on Parallel and Distributed Processing Tech. and Appl.*, 1999. pp. 1097-1103.

[5] M. Handa, R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement", *Proc. Design, Automation Conf.*, 2004. pp. 960-965.

[6] P. C. Vinh, J. P. Bowen, "Continuity Aspects of Embedded Reconfigurable Computing", *Innovations in Systems and Software Eng.: A NASA Journal*, Springer-Verlag, Vol. 1, No. 1, April 2005, pp. 41-53.

[7] A. Ahmadinia, C. Bobda, D. Koch, M. Majer, J. Teich, "Task Scheduling for Heterogeneous Reconfigurable Computers", *Proc. 17th Symp. on Integrated Circuits and Systems Design*, 2004, pp. 22-27.

[8] K. Baskaran, W. Jigang, T. Srikanthan, "Hardware Partitioning Algorithm for Reconfigurable Operating System in Embedded Systems", *Proc. 6th Real-Time Linux Workshop*, 2004, pp. 117-123.

[9] O. Diessel, H. ElGindy, "Run-time Compaction of FPGA Designs", *Proc. 7th Intl. Workshop on Field--Programmable Logic and Appl.*, 1997, pp. 131-140.

[10] A. Ejnioui, R. F. DeMara, "Area Reclamation Strategies and Metrics for SRAM-Based Reconfigurable Devices", *Proc. Intl. Conf. on Eng. of Reconf. Systems and Algorithms*, 2005.

[11] Politécnico di Torino ITC'99 benchmarks. Available at: http://www.cad.polito.it/tools/itc99.html